ExBLAS: Reproducible and Accurate BLAS Library

Roman Ikymchuk†‡, Sylvain Collange‡, David Defour§, Stef Graillat∗
†Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005 Paris, France
‡INRIA – Centre de recherche Rennes – Bretagne Atlantique, Campus de Beaulieu, F-35042 Rennes Cedex, France
§DALI–LIRMM, Université de Perpignan, 52 avenue Paul Alduy, F-66860 Perpignan, France
∗Email: roman.ikymchuk, stef.graillat]@lip6.fr
Email: sylvain.collange@inria.fr
Email: david.defour@univ-perp.fr

Abstract—Due to non-associativity of floating-point operations and dynamic scheduling on parallel architectures, getting a bitwise reproducible floating-point result for multiple executions of the same code on different or even similar parallel architectures is challenging. We address the problem of reproducibility in the context of fundamental linear algebra operations – like the ones included in the BLAS library – and propose algorithms that yield both reproducible and accurate results (correct rounding, except for triangular solver). We present implementations of these algorithms for the BLAS routines along with the performance results in parallel environments such as Intel desktop and server CPUs, Intel Xeon Phi, and both NVIDIA and AMD GPUs.

Keywords—BLAS library, reproducibility, accuracy, superaccumulator, floating-point expansions, error-free transformations, multi- and many-core architectures.

I. INTRODUCTION

Exascale computing ($10^{18}$ operations per second) is likely to be reached within a decade. This increased computational power of modern supercomputers enables us to solve more complex and larger problems. That, consequently, leads to the higher number of floating-point (FP) operations to be performed, each of them potentially causing a round-off error. As FP operations like addition and multiplication are non-associative, so that, for instance, parallel implementations of the BLAS routines become non-reproducible. We define reproducibility as an ability to obtain a bitwise identical FP result from multiple runs of the code on the same input data. However, the result often varies from one parallel machine to another or even from one run to another. These discrepancies worsen on heterogeneous architectures such as clusters with co-processors or GPUs. Such non-determinism of FP calculations in parallel programs causes validation and debugging issues and may even lead to deadlocks. The problem of both non-determinism and non-reproducibility of FP computations becomes so important that it is listed among the top ten ExaScale challenges [5].

Numerical reproducibility can be addressed by targeting either the order of operations or the error resulting from the finite arithmetic. One solution consists in providing the deterministic control over rounding errors by, for example, enforcing the execution order for each operation. However, this approach is not portable and/or does not scale well with the number of processing cores. The other solution aims at avoiding cancellation and rounding errors by using an exact accumulation method such as the superaccumulator proposed by Kulisch [8]. This solution increases the accuracy at the price of more operations and memory transfers. Because of that, for a long time, it was considered too costly.

To enhance reproducibility, Intel proposed a “Conditional Numerical Reproducibility” (CNR) in its MKL. However, CNR does not ensure correct rounding and it induces large performance overhead. For instance, for large arrays the MKL’s summation with CNR is $85\% – 93\%$ slower than both the regular MKL’s and our reproducible summation. Demmel and Nguyen introduced a family of algorithms for reproducible summation in FP arithmetic [3]. Their solution induces a twofold slowdown as data transfers and reductions need to be performed twice. They improved the algorithm [4] using a single reduction among nodes, which minimized the overhead to roughly 20%. Demmel and Nguyen have extended their concept to the other BLAS-1 routines, distributed as ReproBLAS. Arteaga et al. [1] used this approach with improved communication for summation and obtained the same accuracy with roughly 10% overhead. Neal [10] considered scalar superaccumulators of different sizes for summation in the R software. Alternatively, we introduced in [2] a multi-level approach to compute reproducible sums. This approach is based on FP expansions (FPEs) and superaccumulators. The proposed implementations showed that the numerical reproducibility and bit-perfect accuracy can be achieved at no additional cost for large sums with dynamic ranges of up to 90 orders of magnitude.

In this work, we address the problem of reproducibility of the BLAS routines due to the cancellations and rounding errors that occur during FP operations. We begin with parallel reduction using our multi-level summation approach, which allows to perform accumulation in any order without loosing a bit of information. Then, we extend this approach to some of the BLAS routines from level-1 to level-3. The paper is organized as follows. Section [I] reviews aspects of computer arithmetic. Section [II] presents our multi-level reproducible approach. We evaluate our implementations in Section [IV] and discuss conclusions in Section [V].

II. COMPUTER ARITHMETIC

FP arithmetic consists in approximating real numbers with a mantissa, an exponent, and a sign:

$$x = \pm \underbrace{x_0\cdot x_1\ldots x_{M-1}}_{\text{mantissa}} \times b^c, \quad 0 \leq x_i \leq b - 1, \quad x_0 \neq 0$$
where \( b \) is the basis, \( M \) is the precision, and \( e \) is the exponent (
\( e_{\text{min}} \leq e \leq e_{\text{max}} \)). The IEEE-754 standard specifies FP formats and operations as well as requires correctly rounded results for the basic FP operations (\( +, -, \times, /, \sqrt{} \)) and \( \text{fma}^{1} \). This means the operations are performed as if the result was computed with the infinite precision and rounded to the FP format. In this paper, we extend the correct rounding criterion to the BLAS routines. We consider double precision format (binary64) and assume rounding-to-nearest.

Two approaches exist to perform one FP addition without introducing rounding error. The first solution aims at computing the error which occurred during rounding using FP expansions in conjunction with error-free transformations (EFTs). FPEs represent the result as an unevaluated sum of FP numbers, whose components are ordered by magnitude with minimal overlap to cover a wide range of exponents. FPEs of sizes 2 are described in \([9]\). When working with the rounding-to-nearest mode, the rounding error of addition and multiplication can be represented as a FP number. This error can be computed in FP arithmetic using EFTs: \( \text{TwoSum} \), Alg. \([1]\) for addition; \( \text{TwoProd} \), Alg. \([2]\) for multiplication.

**Algorithm 1:** EFT for the sum of two FP numbers.

\[
\text{Function } [r, s] = \text{TwoSum}(a, b) \\
r \leftarrow a + b \\
z \leftarrow r - a \\
s \leftarrow (a - (r - z)) + (b - z)
\]

Adding one FP number to an expansion is an iterative operation. The FP number is first added to the head of the expansion. The rounding error is recovered as a FP number, using an EFT such as \( \text{TwoSum} \), and recursively accumulated to the remainder of the expansion. FPE computations are free of conditional branches and memory accesses, making fast pipelined and vectorized implementations practical. However, their complexity is linear with the number of terms in the FPE, so they are best suited for low dynamic range sums.

The second solution exploits the finite range of representable FP numbers by storing every bit of the sum. It can be implemented as a large fixed-point accumulator (superaccumulator). This accumulator covers the range from the minimum representable FP value to the maximum value independently of the sign. We use a superaccumulator of 2098 (\( e_{\text{min}} + e_{\text{max}} + \text{mantissa} = 1022 + 1023 + 53 \)) bits for double precision FP accumulation. Superaccumulator implementations involve irregular memory accesses, but the amortized complexity of accumulation does not depend on the accumulator length, making it suitable for large dynamic range sums.

**Algorithm 2:** EFT for the product of two FP numbers.

\[
\text{Function } [r, s] = \text{TwoProd}(a, b) \\
r \leftarrow a \times b \\
s \leftarrow \text{fma}(a, b, -r)
\]

Our implementations profit from the technical features of the considered architectures: SIMD instructions, fused-multiply-and-add (FMA) instructions, and multi-threading on CPUs and Xeon Phi; local memory and atomic instructions on GPUs. For example, on the Intel Haswell architecture, we use AVX intrinsics to benefit from the 4-way SIMD and implement \( \text{TwoSum} \) by replacing half of the 6 additions/subtractions by FMAs (multiplying by one). The latter optimization allows to use both the FP adder and FMA units that can run in parallel on Haswell. Thread-level parallelism is exposed using OpenMP to benefit from multi-core and hardware multi-threading. The final inter-node reduction within a cluster is performed using MPI. On the Intel Xeon Phi co-processor, we benefit from 8-way SIMD by using 512-bit vector intrinsics. We perform explicit memory prefetching in order to maximize memory throughput. We develop hand-tuned OpenCL implementations for NVIDIA and AMD GPUs. They use multiple superaccumulators per work group, which are stored in local or global memory. In order to avoid bank conflicts, superaccumulators are interleaved together to spread their digits among different memory banks. Concurrency between multiple threads, which share single superaccumulator, is handled through atomics.

As baselines, we consider our vectorized and parallelized non-deterministic algorithms. In case of summation, we also

\[\text{fma}(a, b, c) \text{ computes } a \times b + c \text{ with a single rounding.}\]
use: the deterministic Intel TBB reduction (referred as “TBB deterministic”); the Fast Deterministic Parallel Sum algorithm \[3\] (“Demmel fast”); the one reduction reproducible summation \[4\] (“ReproBLAS”) from the ReproBLAS library; the single-sweep reduction \[11\] with two and three levels (“bitrep2” and “bitrep3”, accordingly) from the bitrep library.

Figs. 1a and 1b present the timings achieved by the summation algorithms as a function of the input dataset size \(N\) of double precision FP numbers. In the legends of all figures, “Superacc” corresponds to our algorithm that is solely based on superaccumulators; “FPE\(p\) + Superacc” stands for our algorithm with FPEs of size \(p\) (\(p = 2 : 8\)) in conjunction with superaccumulators when needed; “FPE\(p\)EE + Superacc” represents our algorithm based on the expansion of size \(p\) and the early-exit optimization technique, meaning stop propagating errors when they are zeros. These plots show that on large datasets the performance of the baseline unordered sum as well as the expansion cache is constrained by the memory bandwidth on all platforms. For instance, these algorithms achieve 23.3 GB/s on platform A over the theoretical peak bandwidth of 25.6 GB/s. As expected, for small datasets that fits in cache, the non-reproducible parallel sum outperforms our implementations by a factor of 2 to 5. “TBB deterministic” and MPFR (not shown on the plot) run two and three orders of magnitude, respectively, slower than our implementations. The Fast Deterministic Parallel Sum algorithm requires two passes on the data. In memory-bounded scenarios, its execution is two times slower than the unordered reduction and the other single-pass algorithms. In addition, the overhead of the second kernel call impacts the performance of the GPU implementation.

Fig. 2 compares the performance scaling of the reduction algorithms on platform B. The number of MPI process varies from 1 to 64, each of them performing the summation of 16M double-precision FP numbers. This dataset size ensure that we fall in the out-of-cache case. For each process, we measure the local summation time, which is colored red, and the MPI reduction time, which is colored green. For the whole range of processors, the execution time of each algorithm is dominated by the local summation time because of the dataset size. In addition, due to the equal distribution of computations among MPI processes, the computation time is roughly equivalent on the whole range of MPI processes, while the reduction time changes according to the number of MPI processes involved. We normalize the total runtime of each algorithm by the total execution time of the parallel FP summation. The “bitrep2”, “bitrep3”, “Demmel fast”, and “ReproBLAS” algorithms are 2.9\%, 3.1\%, 95.5\%, and 20.6\%, respectively, slower than the conventional parallel summation on 64 CPUs. In contrast, our “FPE2 + Superacc” and “FPE\(p\)EE + Superacc” deliver both correctly rounded and bit-wise reproducible results with the overhead of 9.2\% and 7.8\%, respectively.

We extend the multi-level summation algorithm to dot product by applying TwoProd for exact multiplication of two FP numbers. Fig. 1c shows the performance results of dot product on platform D, see Tab. 1 TwoProd induces small performance overhead due to summation of two arrays: one for the result and another for the error. Therefore, the performance penalties are slightly higher than in case of ExSUM. Nevertheless, for large array sizes, ExDOT delivers numerically reproducible results with no performance loses.

On the BLAS level-2, we propose a blocked multi-level reproducible algorithm for triangular solver (ExTRSV). The idea behind is to divide matrix by blocks and apply ExTRSV on diagonal blocks and matrix-vector multiplication (ExGEMV) on sub/upper diagonal blocks; the later is benefi-
We presented a multi-level approach that delivers both accurate and bit-wise reproducible results for fundamental linear algebra operations such as the ones included in the BLAS library. We derived an algorithm for parallel FP reduction, which is a common operation among the BLAS routines. More information on the ExBLAS project as well as its sources can be found at https://exblas.lip6.fr/.

ACKNOWLEDGEMENT

This work undertaken in the framework of CALSIMLAB is supported by the public grant ANR-11-LABX-0037-01 and used the HPC resources of ICS funded by Region Île-de-France and the project Equip@Meso (ANR-10-EQPX-29-01) both overseen by the French National Research Agency (ANR) as part of the “Investissements d’Avenir” program (ANR-11-IDEX-0004-02). This work was partially supported by the project FastRelax (ANR-14-CE25-0018-01).

REFERENCES


